

# **Apple Stock Price Prediction**

## **Business Problem:**

In the financial markets, predicting stock prices is a critical challenge faced by the investors and financial institutions. The main business problem addressed by this project is the uncertainty around the Apple Inc.'s stock prices. Investors often depend on forecasts to make well-informed decisions about trading the stocks, managing portfolios, optimizing investments strategies and many more. The unpredictable nature of stock market needs sophisticated models which can capture intricate patterns and trend from the historical data. This project aims to provide a robust solution to the business problem by leveraging Machine Learning techniques, especially Random Forest Regressor, to create an accurate and reliable predictor of Apple stock prices. The successful implementation of the predictive model can empower stakeholders with valuable insights, aiding them in navigating the complexities of stock market and enhancing their ability to make informed financial decisions.

## **Objectives:**

The primary objective of this project is to develop a robust and accurate predictive model for Apple Ins.'s stock prices. By leveraging historical stock data and utilizing Machine Learning algorithms, we aim to create a tool that can forecast future stock prices with a high degree of precision. This predictive model will serve as a valuable resource for investors, financial analysts, and other stakeholders and provide them with insights that can inform their decision-making processes.

Some of the key objectives are to

- perform an in-depth analysis of the key features influencing Apple stock prices, such as opening price, closing price, high and low prices, adjusted close prices, and trading volumes.
- Explore and visualize historical stock data to identify patterns, trends and potentials relationships between different features and stock prices.
- Evaluate and compare various Machine Learning models, such as Linear Regression, Random Forest, XGBoost, and others, to determine the most suitable model for predicting stock prices.

- Assess the performance of the selected models using appropriate metrics such as R-squared, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).
- Develop a user-friendly web application using Streamlit, allowing users to input relevant stock data and obtain real-time predictions for Apple stock prices.

By achieving these objectives, this project aims to provide a solution that combines technical excellence with practical usability.

### **Solution Approach:**

The solution commences with acquisition of historical stock data for Apple Inc., sourced from the "AAPL.csv" dataset. This dataset encompasses essential features, including opening price, closing price, high and low prices, adjusted close prices, and trading volumes, spanning several decades.

Upon data collection, the dataset undergoes thorough preprocessing to handle missing values, if any, and ensure the consistency and integrity of the data. Exploratory Data Analysis (EDA) techniques are employed to visualize and understand the relationships between different features and the target variable stock prices.

The core of the solution involves the development of a predictive model for Apple stock prices. Multiple Machine Learning algorithms, including Linear Regression, Random Forest, XGBoost, and others, are explored and evaluated. The model is trained on historical data to learn the underlying patterns and relationships, with a focus on achieving high accuracy and reliability in predicting future stock prices.

The performance of each model is rigorously evaluated using metrics such as R-squared, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). This evaluation process guides the selection of the most suitable model for predicting Apple stock prices, ensuring that the chosen model aligns with the project's objectives.

To enhance user accessibility, a user-friendly web application is developed using Streamlit. This application allows users to input relevant stock data and obtain real-time predictions for Apple stock prices. The deployment of the application ensures that stakeholders can effortlessly interact with the predictive model, making informed decisions based on the latest stock price forecasts.

By following this comprehensive solution approach, the project aims to deliver a sophisticated yet user-friendly tool that empowers stakeholders with accurate and timely predictions for Apple Inc.'s stock prices.

### **Scope:**

The scope of the Apple Stock Price Prediction project is defined by its ability to provide valuable insights into future stock prices for Apple Inc. This project primarily focuses on historical stock data analysis and employs machine learning models to forecast stock prices based on various features. The scope encompasses the development of a user-friendly web application, allowing stakeholders to interact with the predictive model effortlessly. The application accepts user inputs such as date, opening price, high and low prices, adjusted close prices, and volume, providing real-time predictions for the closing price of Apple stock.

The project's scope extends to financial analysts, investors, and anyone interested in making informed decisions regarding Apple Inc.'s stock. The predictive models developed aim to capture intricate patterns and trends within the historical data, enabling users to anticipate potential fluctuations in stock prices. However, it's essential to note that the predictions are based on historical patterns and do not account for unforeseen events or market dynamics that may influence stock prices.

The project's scope also includes ongoing model updates and improvements to enhance prediction accuracy and align with changing market conditions. While the project is centered around Apple Inc.'s stock, the methodology and tools developed can potentially be extended to analyze and predict stock prices for other companies, broadening the project's applicability in the financial domain.

### **Team Size:**

Our team consists of 5 members. They are:

- M Hemanth Kumar Yadav
- G Chaitanya
- Sai Sri Bhavna
- Jyothirmayee
- Sai Siva Krishna

## **Data Sources:**

The dataset used in this project is taken from the Kaggle website. The dataset is regarding the stock prices of Apple Inc.'s stock prices. It contains columns like "Date", "Open", "High", "Low", "Close", "Adj Close", "Volume".

Link for the Dataset: <https://www.kaggle.com/datasets/meetnagadia/apple-stock-price-from-19802021>

## **Data Preparation:**

Data preparation is a critical phase in the Apple Stock Price Prediction project, involving several key steps to ensure the dataset is well-suited for model training and analysis. The initial step includes loading the historical stock data from the "AAPL.csv" dataset using the Pandas library in Python. This dataset contains multiple features, including opening and closing prices, high and low prices, adjusted close prices, and trading volumes.

Exploratory Data Analysis (EDA) is conducted to gain insights into the dataset's structure and identify potential patterns. The EDA phase involves checking the dataset's shape, data types, and statistical summaries to understand the distribution of each feature. Visualization tools such as Matplotlib and Seaborn are employed to create plots and charts, aiding in the interpretation of stock price trends over time.

Handling missing values is a crucial aspect of data preparation, ensuring the dataset's completeness. Date columns are converted to a consistent format, allowing for seamless integration with time-series analysis techniques. Normalization and scaling are applied to numerical features to bring them to a comparable scale, preventing certain features from dominating the model training process.

The prepared dataset serves as the foundation for building and training machine learning models. Rigorous data preparation ensures that the models can effectively learn from the historical stock data, leading to accurate and reliable predictions of Apple's future stock prices.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv(r"E:\Apple Stock Price Prediction\AAPL.csv")
df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

```
df.shape
```

```
(10468, 7)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10468 entries, 0 to 10467
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  -
 0   Date        10468 non-null object  
 1   Open        10468 non-null float64 
 2   High        10468 non-null float64 
 3   Low         10468 non-null float64 
 4   Close       10468 non-null float64 
 5   Adj Close   10468 non-null float64 
 6   Volume      10468 non-null int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 572.6+ KB
```

```
df.isnull().any()
```

```
Date      False
Open      False
High      False
Low       False
Close     False
Adj Close False
Volume    False
dtype: bool
```

```
df.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000	1.046800e+04
mean	14.757987	14.921491	14.594484	14.763533	14.130431	3.308489e+08
std	31.914174	32.289158	31.543959	31.929489	31.637275	3.388418e+08
min	0.049665	0.049665	0.049107	0.049107	0.038329	0.000000e+00
25%	0.283482	0.289286	0.276786	0.283482	0.235462	1.237768e+08
50%	0.474107	0.482768	0.465960	0.475446	0.392373	2.181592e+08
75%	14.953303	15.057143	14.692589	14.901964	12.835269	4.105794e+08
max	182.630005	182.940002	179.119995	182.009995	181.511703	7.421641e+09

```
df.corr()
```

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999943	0.999924	0.999850	0.999516	-0.196211
High	0.999943	1.000000	0.999908	0.999924	0.999618	-0.195635
Low	0.999924	0.999908	1.000000	0.999928	0.999575	-0.197150
Close	0.999850	0.999924	0.999928	1.000000	0.999671	-0.196411
Adj Close	0.999516	0.999618	0.999575	0.999671	1.000000	-0.199262
Volume	-0.196211	-0.195635	-0.197150	-0.196411	-0.199262	1.000000

```
print("Correlations with 'Volume' column:")
df.corr().Volume.sort_values(ascending=False)
```

```
Correlations with 'Volume' column:
```

```
Volume      1.000000  
High        -0.195635  
Open        -0.196211  
Close       -0.196411  
Low         -0.197150  
Adj Close   -0.199262  
Name: Volume, dtype: float64
```

## Data Visualization:

Data visualization plays a pivotal role in the Apple Stock Price Prediction project, providing a visual representation of the historical stock data. Various visualization libraries such as Matplotlib and Seaborn are utilized to create insightful plots and charts that aid in understanding trends and patterns.

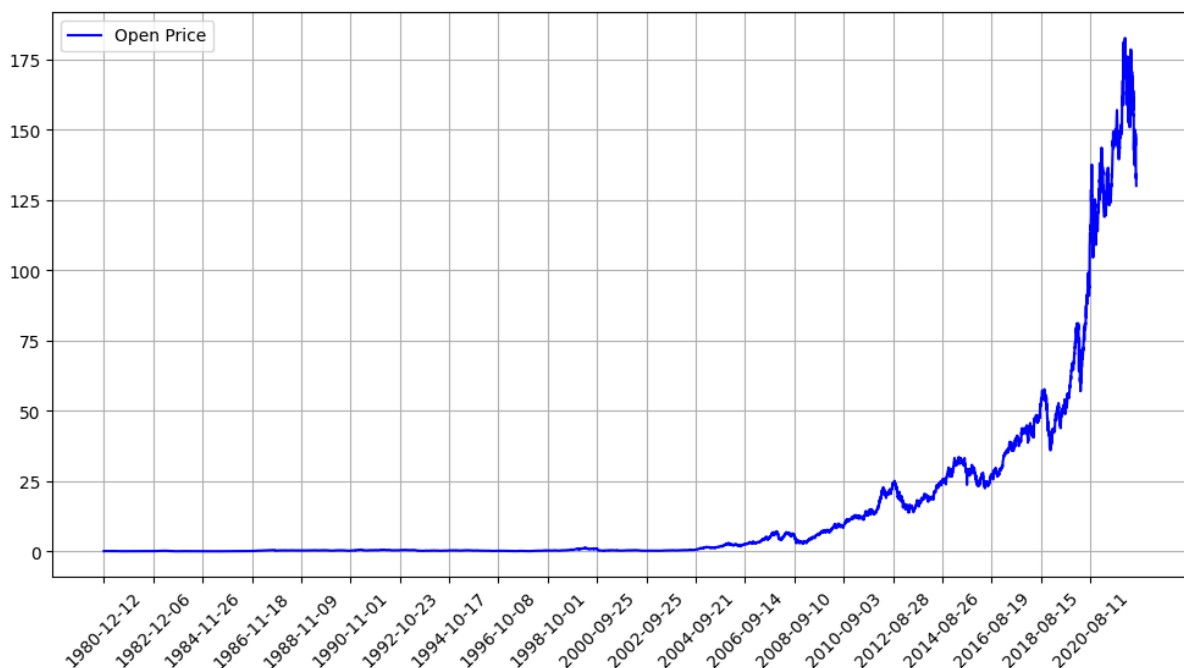
Plots illustrate the opening, closing, high, and low prices over the dataset's time span, enabling the identification of long-term and short-term trends. Scatter plots and bar charts depict relationships between different features, offering a comprehensive view of the dataset's structure.

Histograms and distribution plots are employed to explore the distribution of stock prices and trading volumes, helping in identifying outliers and skewed patterns. Correlation matrices showcase the relationships between different features, assisting in the identification of potential predictors for stock price movements.

The visualizations not only serve as a means of exploratory analysis but also aid in communicating findings to stakeholders effectively. Through data visualization, complex patterns become accessible, fostering a deeper understanding of the underlying dynamics of Apple's stock prices.

Code:

```
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Open'], label='Open Price', color='b')
plt.title = ('Apple Stock Open Price Over Time')
plt.xlabel = ('Date')
plt.ylabel = ('Open Price')
plt.xticks(df['Date'][:500], rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```



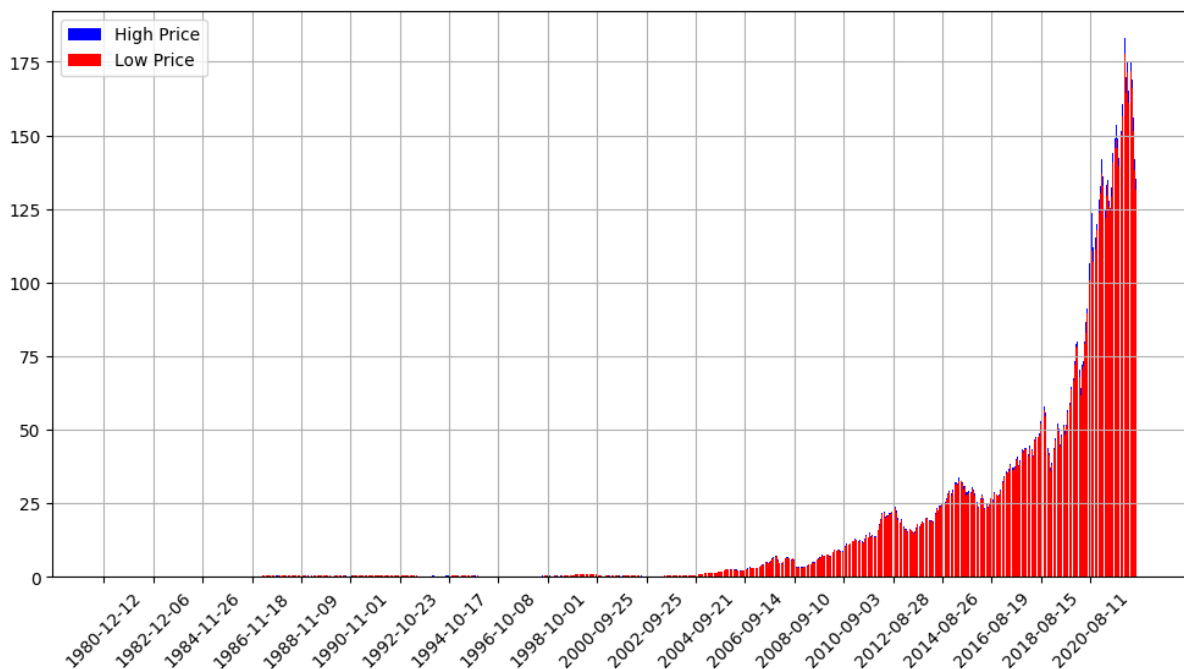
Conclusion from above Chart:

The above plot shows the relation between 'Open' price and 'Date' columns.

Here we can see that Date and Open columns are linearly related over a long time.

```
plt.figure(figsize=(12, 6))
plt.bar(df['Date'], df['High'], label='High Price', color='b')
plt.bar(df['Date'], df['Low'], label='Low Price', color='r')
plt.title = ('Apple Stock High and Low Prices Over Time')
plt.xlabel = ('Date')
plt.ylabel = ('High and Low Prices')
plt.xticks(df['Date'][:500], rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```





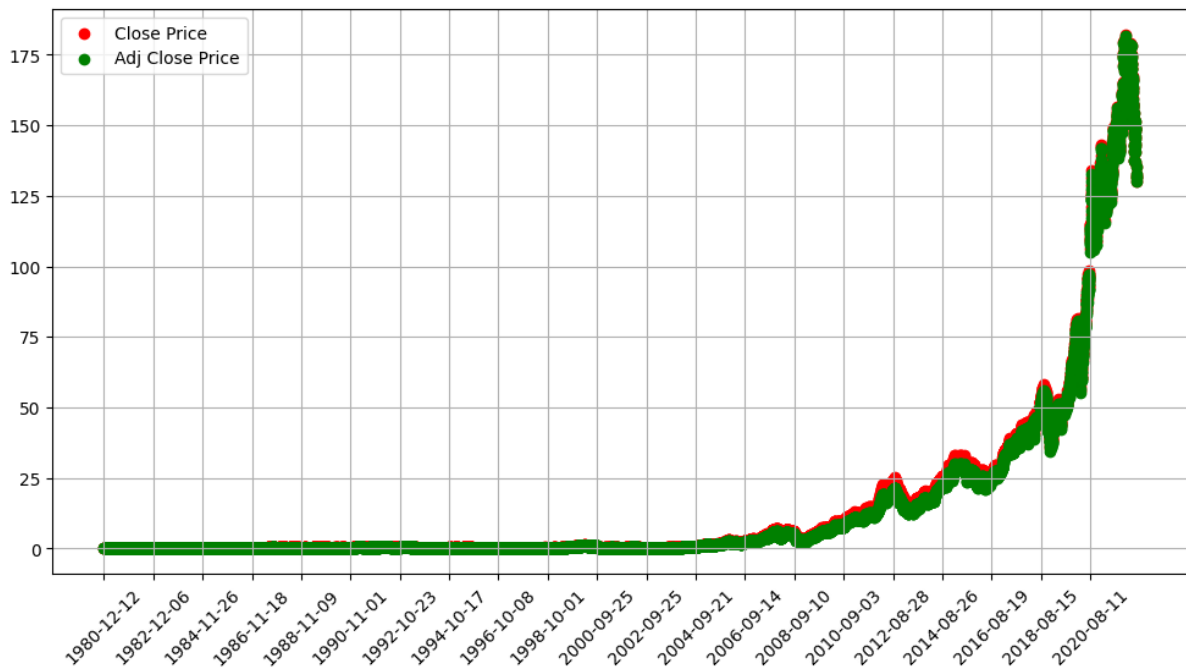
Conclusion from above Chart:

The above plot shows the relation between 'Low', 'High' prices and 'Date' columns.

Here we can see that Low and High do not have a large variation in values.

And Low and High columns are linearly related to Date over a long range of time.

```
plt.figure(figsize=(12, 6))
plt.scatter(df['Date'], df['Close'], label='Close Price', color='r')
plt.scatter(df['Date'], df['Adj Close'], label='Adj Close Price', color='g')
plt.title = ('Apple Stock Close and Adj Close Prices Over Time')
plt.xlabel = ('Date')
plt.ylabel = ('Close and Adj Close Prices')
plt.xticks(df['Date'][:500], rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```



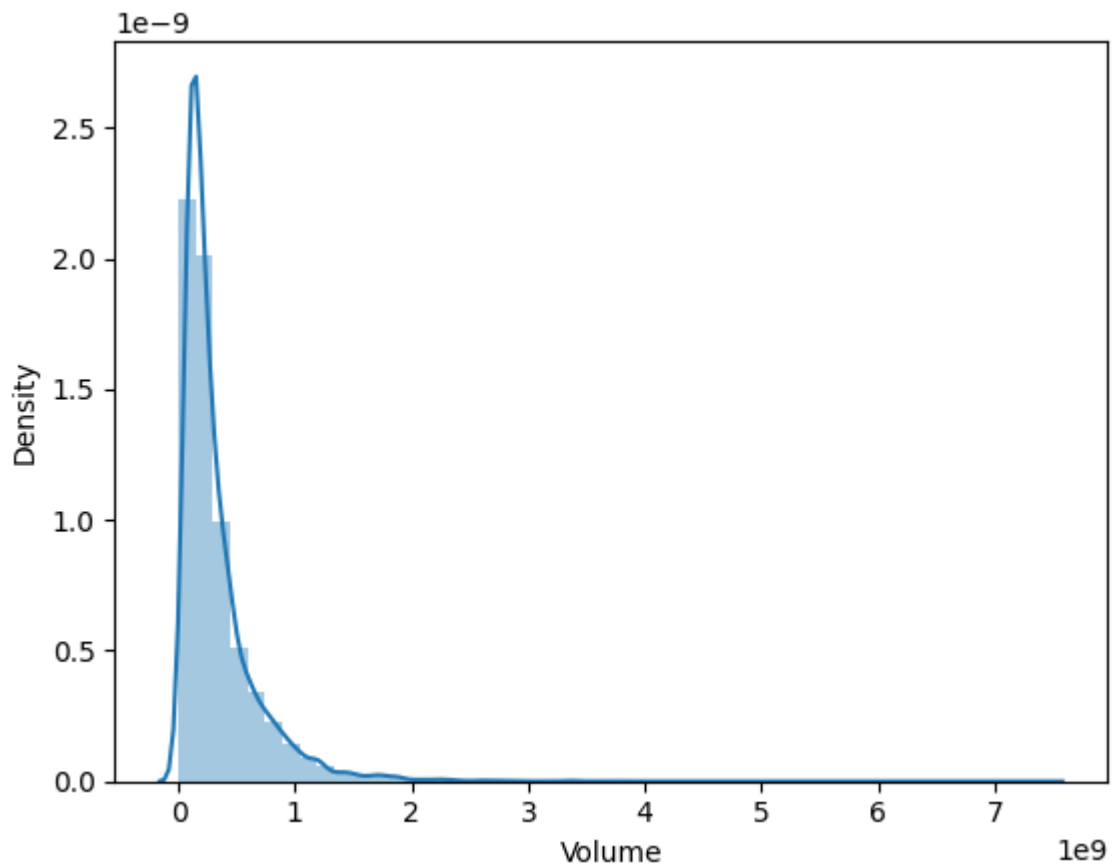
Conclusion from above Chart:

The above plot shows the relation between 'Close', 'Adj Close' prices and 'Date' columns.

Here we can see that Close and Adj Close were maximum at the year of 2021.

There has been a decreasing trend since then.

```
sns.distplot(df.Volume)
```

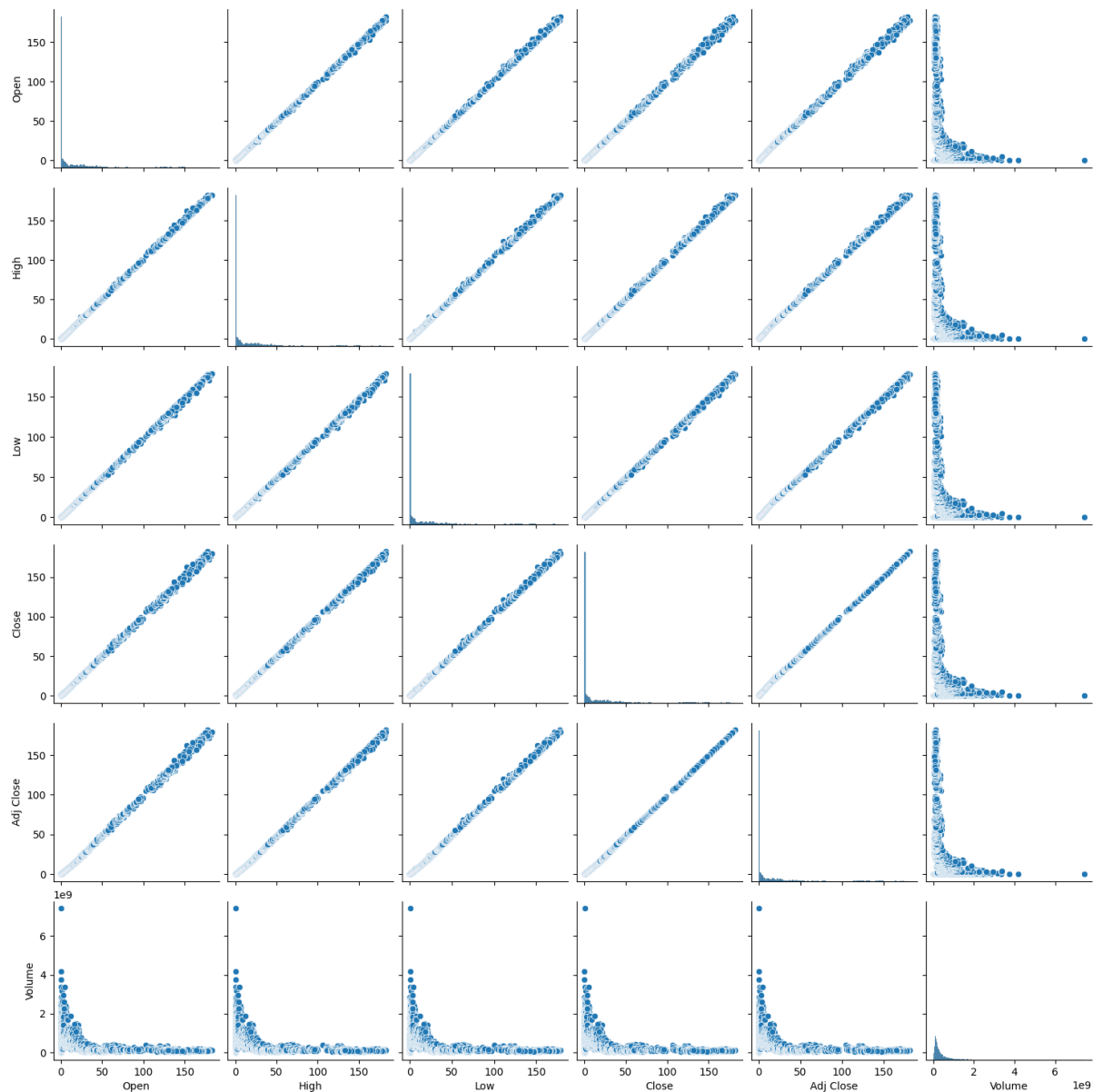


Conclusion from above Chart

The above plot shows the Distribution of the 'Volume'.

It has a skewed distribution towards right and hence follows Right Skewed Distribution or Positive skewness.

```
sns.pairplot(df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']])  
plt.show()
```



Conclusion from above Chart:

The above plot shows the Pair Plot of all the numerical columns.

It displays scatterplots of each variable plotted against every other variable.

The plots which are diagonal from the top left to bottom right represent the histograms of each variable.

The plots which are other places represent the scatter plots of pair of variables which explain the relationship between them.

## Statistical Analysis:

Statistical analysis is a critical aspect of the Apple Stock Price Prediction project, providing a quantitative understanding of the dataset's characteristics. Descriptive statistics, including mean, median, mode, and measures of dispersion, offer insights into the central tendencies and variability of the stock price and trading volume data.

Correlation matrices unveil the strength and direction of relationships between different features, aiding in feature selection for the predictive models. The analysis of distribution plots helps identify the underlying distribution patterns, such as normality or skewness, which informs the choice of appropriate machine learning algorithms.

Moreover, statistical tests may be employed to validate assumptions and draw meaningful conclusions about the dataset. Through statistical analysis, the project aims to uncover hidden patterns, assess the reliability of the data, and make informed decisions regarding the model's architecture and predictive capabilities.

Code:

```
#Descriptive Statistics
summary_stats = df.describe()
print("Descriptive Statistics: ")
print(summary_stats)
```

### Descriptive Statistics:

	Open	High	Low	Close	Adj Close	\
count	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000	
mean	14.757987	14.921491	14.594484	14.763533	14.130431	
std	31.914174	32.289158	31.543959	31.929489	31.637275	
min	0.049665	0.049665	0.049107	0.049107	0.038329	
25%	0.283482	0.289286	0.276786	0.283482	0.235462	
50%	0.474107	0.482768	0.465960	0.475446	0.392373	
75%	14.953303	15.057143	14.692589	14.901964	12.835269	
max	182.630005	182.940002	179.119995	182.009995	181.511703	

	Volume
count	1.046800e+04
mean	3.308489e+08
std	3.388418e+08
min	0.000000e+00
25%	1.237768e+08
50%	2.181592e+08
75%	4.105794e+08
max	7.421641e+09

### #Correlation Analysis

```
correlation_matrix = df[['Open', 'High', 'Low', 'Close', 'Adj Close',
'Volume']].corr()
print("Correlation Matrix: ")
print(correlation_matrix)
```

### Correlation Matrix:

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999943	0.999924	0.999850	0.999516	-0.196211
High	0.999943	1.000000	0.999908	0.999924	0.999618	-0.195635
Low	0.999924	0.999908	1.000000	0.999928	0.999575	-0.197150
Close	0.999850	0.999924	0.999928	1.000000	0.999671	-0.196411
Adj Close	0.999516	0.999618	0.999575	0.999671	1.000000	-0.199262
Volume	-0.196211	-0.195635	-0.197150	-0.196411	-0.199262	1.000000

### #Central Tendency

```
mean_open = df['Open'].mean()
median_open = df['Open'].median()
mode_open = df['Open'].mode().values[0]
print(f"Mean Open Price: {mean_open}")
print(f"Median Open Price: {median_open}")
print(f"Mode Open Price: {mode_open}")
print()
mean_high = df['High'].mean()
median_high = df['High'].median()
```

```
mode_high = df['High'].mode().values[0]
print(f"Mean High Price: {mean_high}")
print(f"Median High Price: {median_high}")
print(f"Mode High Price: {mode_high}")
print()
low_values = df['Low']
mean_low = low_values.mean()
median_low = low_values.median()
mode_low = low_values.mode().iloc[0]
print(f"Mean Low: {mean_low}")
print(f"Median Low: {median_low}")
print(f"Mode Low: {mode_low}")
print()
close_mean = df['Close'].mean()
close_median = df['Close'].median()
close_mode = df['Close'].mode().iloc[0]
print(f"Mean Close: {close_mean}")
print(f"Median Close: {close_median}")
print(f"Mode Close: {close_mode}\n")

adj_close_mean = df['Adj Close'].mean()
adj_close_median = df['Adj Close'].median()
adj_close_mode = df['Adj Close'].mode().iloc[0]
print(f"Mean Adj Close: {adj_close_mean}")
print(f"Median Adj Close: {adj_close_median}")
print(f"Mode Adj Close: {adj_close_mode}\n")

volume_mean = df['Volume'].mean()
volume_median = df['Volume'].median()
volume_mode = df['Volume'].mode().iloc[0]
print(f"Mean Volume: {volume_mean}")
print(f"Median Volume: {volume_median}")
print(f"Mode Volume: {volume_mode}")
```

```
Mean Open Price: 14.757987364444018
Median Open Price: 0.474107
Mode Open Price: 0.354911
```

```
Mean High Price: 14.92149111683225
Median High Price: 0.4827675
Mode High Price: 0.372768
```

```
Mean Low: 14.594484334925486
Median Low: 0.46596
Mode Low: 0.357143
```

```
Mean Close: 14.76353296264807
Median Close: 0.475446
Mode Close: 0.399554
```

```
Mean Adj Close: 14.130431202044326
Median Adj Close: 0.392373
Mode Adj Close: 0.086241
```

```
Mean Volume: 330848870.0993504
Median Volume: 218159200.0
Mode Volume: 246400000
```

```
#Z - Score
column_name = 'Open'
data_column = df[column_name]
mean = data_column.mean()
std_dev = data_column.std()
data_point = 150.0
z_score = (data_point - mean) / std_dev
print(f"Z-Score for {data_point} in the '{column_name}' column:
{z_score:.4f}")
```

```
Z-Score for 150.0 in the 'Open' column: 4.2377
```

## Exploratory Data Analysis:

Exploratory Data Analysis (EDA) is a crucial phase in the Apple Stock Price Prediction project, involving the examination and visualization of the dataset to uncover patterns, trends, and potential outliers. Utilizing libraries like Pandas, NumPy, Matplotlib, and Seaborn, the EDA process includes summary statistics, distribution plots, correlation matrices, and pair plots.



The analysis of stock price trends over time, as depicted in line charts, bar graphs, and scatter plots, aids in understanding the relationships between different features. EDA also involves investigating the distribution of stock prices, trading volumes, and other relevant variables, providing valuable insights into their statistical properties.

Through EDA, the project aims to gain a comprehensive understanding of the dataset's characteristics, identify any anomalies or outliers, and make informed decisions about feature engineering and model selection. The visualizations generated during EDA serve as a foundation for subsequent steps, guiding the development of machine learning models for accurate stock price predictions.

Code:

```
print("Dataset Info:")
df.info()
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10468 entries, 0 to 10467
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   Date        10468 non-null  object
 1   Open        10468 non-null  float64
 2   High        10468 non-null  float64
 3   Low         10468 non-null  float64
 4   Close       10468 non-null  float64
 5   Adj Close   10468 non-null  float64
 6   Volume      10468 non-null  int64
dtypes: float64(5), int64(1), object(1)
memory usage: 572.6+ KB
```

```
print("Dataset Statistics:")
print(df.describe())
```

### Dataset Statistics:

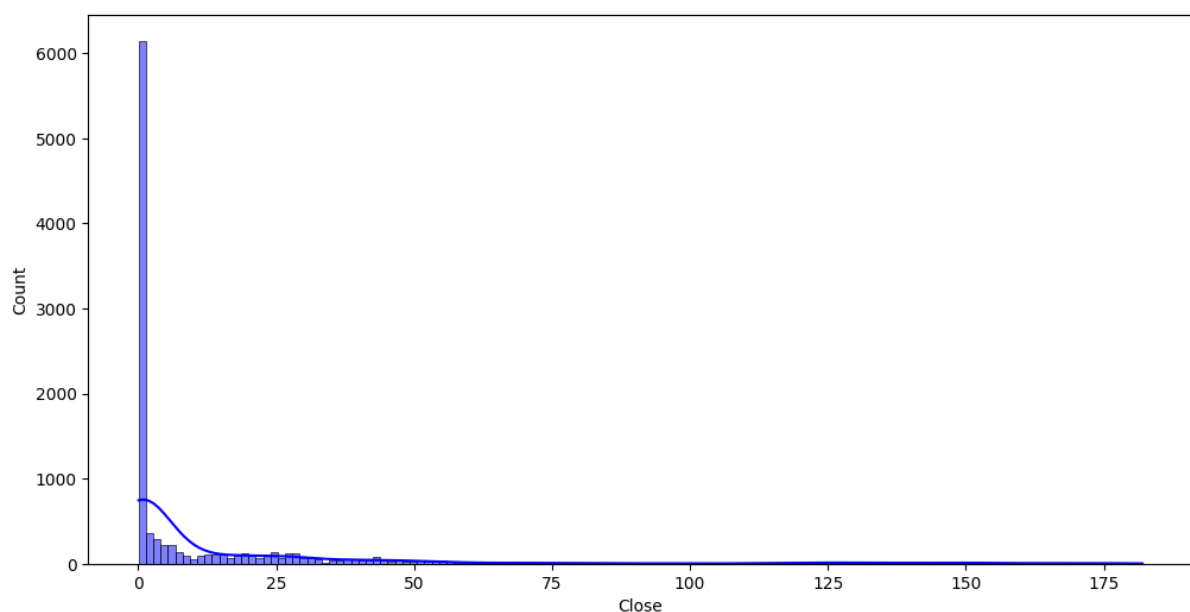
	Open	High	Low	Close	Adj Close \
count	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000
mean	14.757987	14.921491	14.594484	14.763533	14.130431
std	31.914174	32.289158	31.543959	31.929489	31.637275
min	0.049665	0.049665	0.049107	0.049107	0.038329
25%	0.283482	0.289286	0.276786	0.283482	0.235462
50%	0.474107	0.482768	0.465960	0.475446	0.392373
75%	14.953303	15.057143	14.692589	14.901964	12.835269
max	182.630005	182.940002	179.119995	182.009995	181.511703

### Volume

count	1.046800e+04
mean	3.308489e+08
std	3.388418e+08
min	0.000000e+00
25%	1.237768e+08
50%	2.181592e+08
75%	4.105794e+08
max	7.421641e+09

### #Data Distributions

```
plt.figure(figsize=(12, 6))
sns.histplot(df['Close'], kde=True, color='blue')
plt.xlabel = ('Close Price')
plt.ylabel = ('Frequency')
plt.title = ('Distribution of Close Prices')
plt.show()
```



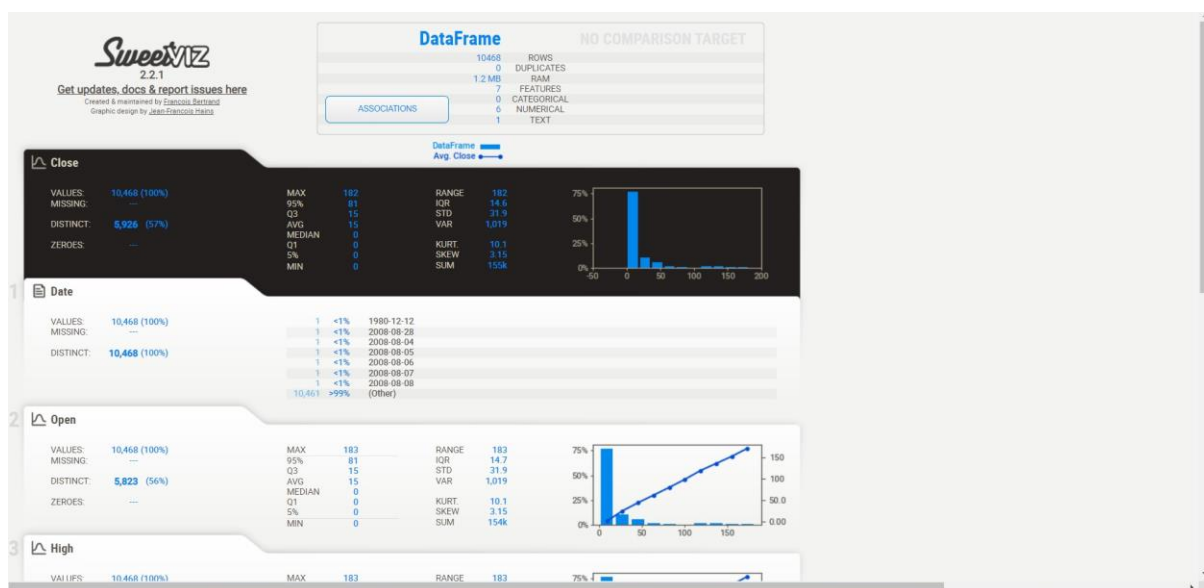
Conclusion from above Chart:

The above plot shows the Distribution of the 'Close'.

It has a skewed distribution towards right and hence follows Right Skewed Distribution or Positive skewness.

The mode of 'Close' price is near 0 i.e. most of the Close prices occur near 0.

```
import sweetviz as sv
report = sv.analyze(df, target_feat='Close')
report.show_html('sweetviz_report.html')
```



Models:

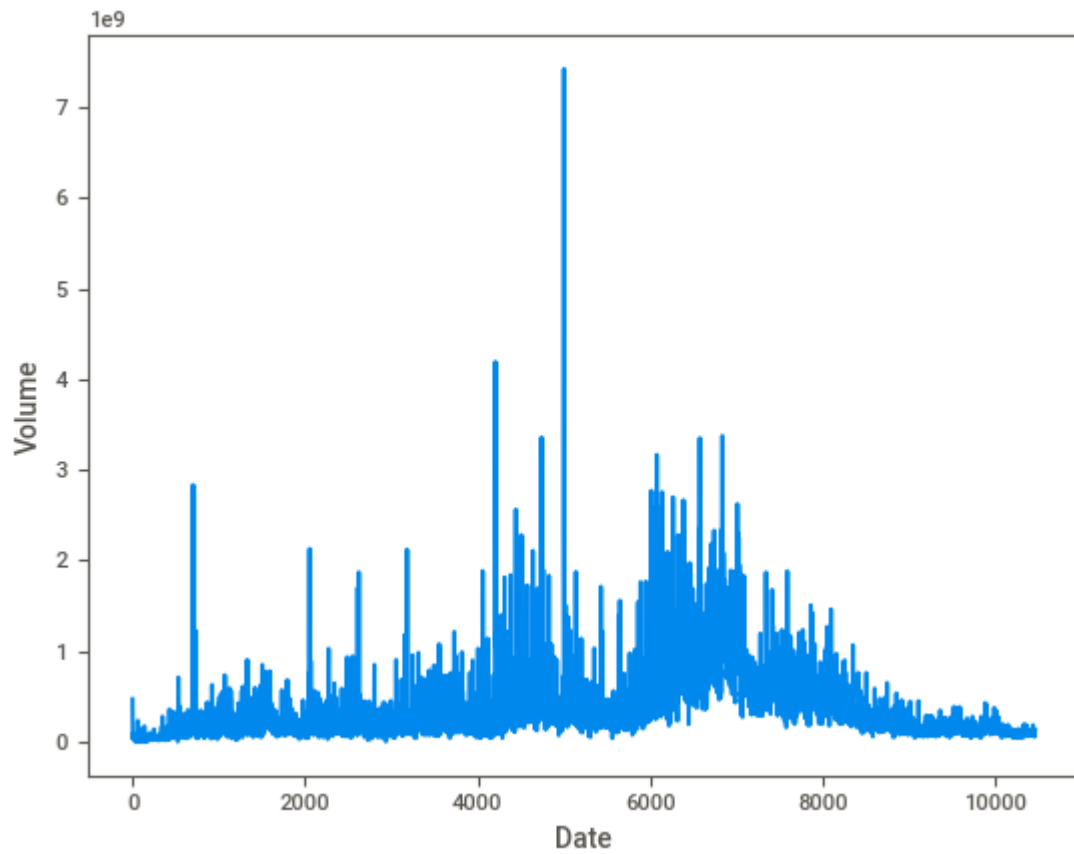
Code:

```
from sklearn.preprocessing import LabelEncoder
```

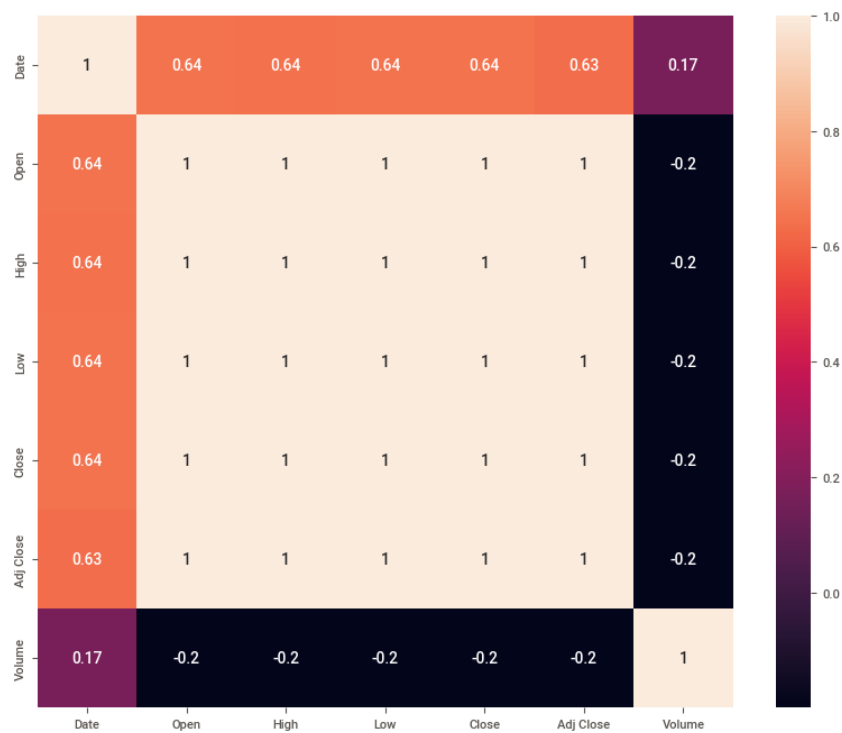
```
le = LabelEncoder()
```

```
df.Date = le.fit_transform(df.Date)
```

```
sns.lineplot(x = df.Date, y=df.Volume)
```



```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),annot =True)
```



```
df.Volume.value_counts()
```

```
246400000    7
239680000    6
244160000    6
302400000    5
255360000    5
..
260915200    1
244652800    1
209171200    1
119470400    1
134118500    1
Name: Volume, Length: 9905, dtype: int64
```

```
y=df.Close
X = df.drop(columns=['Close'],axis=1)
X.head()
```

	Date	Open	High	Low	Adj Close	Volume
0	0	0.128348	0.128906	0.128348	0.100178	469033600
1	1	0.122210	0.122210	0.121652	0.094952	175884800
2	2	0.113281	0.113281	0.112723	0.087983	105728000
3	3	0.115513	0.116071	0.115513	0.090160	86441600
4	4	0.118862	0.119420	0.118862	0.092774	73449600

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size =
0.3,random_state = 47)
```

```
from sklearn.linear_model import LinearRegression
LR_model = LinearRegression()
LR_model.fit(x_train,y_train)
```

```
LR_y_predict=LR_model.predict(x_test)
LR_y_predict
LR_y_predict1=LR_model.predict(x_train)
LR_y_predict1
```

```
array([ 55.0448546 , 149.249849 ,  0.32123438, ...,  4.61183011,
        0.36106384,  0.40313433])
```

```
from sklearn import metrics

# R- Square
# evaluating testing accuracy
print("Linear Regression:")
LR_y_pred_r2 = (metrics.r2_score(y_test,LR_y_predict))
print("R-Squared for x_test: ",LR_y_pred_r2)

# evaluating traing accuracy
LR_y_pred1_r2 = (metrics.r2_score(y_train,LR_y_predict1))
print("R-Squared for x_train: ",LR_y_pred1_r2)

# MSE (Mean square Error)
LR_y_pred_mse = metrics.mean_squared_error(y_test,LR_y_predict)
print("Mean Squared Error for x_test: ",LR_y_pred_mse)

# RMSE (Root Mean Square Error)
LR_y_pred_rmse = np.sqrt(metrics.mean_squared_error(y_test,LR_y_predict))
print("Root Mean Squared Error for x_test: ",LR_y_pred_rmse)
```

```
Linear Regression:
R-Squared for x_test:  0.9999449461013432
R-Squared for x_train:  0.9999451219081523
Mean Squared Error for x_test:  0.05046307961001372
Root Mean Squared Error for x_test:  0.22463988873308702
```

```
from sklearn.ensemble import RandomForestRegressor
RF_model = RandomForestRegressor(n_estimators=100, random_state=42)
RF_model.fit(x_train,y_train)
```

```
RF_y_pred = RF_model.predict(x_test)
RF_y_pred
```

```
array([ 0.53648923,  0.43041824,  0.19342057, ..., 170.5386004 ,
        0.37498958,  0.45620685])
```

```
print("Random Forest Regressor:")
RF_y_pred_r2 = (metrics.r2_score(y_test,RF_y_pred))
print("R-Squared for x_test: ",RF_y_pred_r2)

RF_y_pred_mse = metrics.mean_squared_error(y_test,RF_y_pred)
print("Mean Squared Error for x_test: ",RF_y_pred_mse)

RF_y_pred_rmse = np.sqrt(metrics.mean_squared_error(y_test,RF_y_pred))
print("Root Mean Squared Error for x_test: ",RF_y_pred_rmse)
```

```
Random Forest Regressor:
R-Squared for x_test:  0.9999812803242103
Mean Squared Error for x_test:  0.017158684719811617
Root Mean Squared Error for x_test:  0.1309911627546363
```

```
from xgboost import XGBRegressor
XGB_model =
XGBRegressor(n_estimators=100,learning_rate=0.2,max_depth=12,random_state=1)
XGB_model.fit(x_train,y_train)
```

```
XGB_y_pred = XGB_model.predict(x_test)
XGB_y_pred
```

```
array([ 0.546294 ,  0.4287685 ,  0.18922019, ..., 167.16023 ,
        0.37143144,  0.45937356], dtype=float32)
```

```
print("XGB Regressor")
XGB_y_pred_r2 = (metrics.r2_score(y_test,XGB_y_pred))
print("R-Squared for x_test: ",XGB_y_pred_r2)

XGB_y_pred_mse = metrics.mean_squared_error(y_test,XGB_y_pred)
print("Mean Squared Error for x_test: ",XGB_y_pred_mse)

XGB_y_pred_rmse = np.sqrt(metrics.mean_squared_error(y_test,XGB_y_pred))
print("Root Mean Squared Error for x_test: ",XGB_y_pred_rmse)
```

```
XGB Regressor
R-Squared for x_test:  0.9996881239020564
Mean Squared Error for x_test:  0.28586946143647746
Root Mean Squared Error for x_test:  0.5346676177182208
```

```
from sklearn.ensemble import GradientBoostingRegressor
GBR_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.2,
max_depth=3, random_state=42)
GBR_model.fit(x_train, y_train)
```

```
GBR_y_predict = GBR_model.predict(x_test)
```

```
GBR_y_pred_r2 = metrics.r2_score(y_test, GBR_y_predict)
GBR_y_pred_mse = metrics.mean_squared_error(y_test, GBR_y_predict)
GBR_y_pred_rmse = np.sqrt(GBR_y_pred_mse)
```

```
print("Gradient Boosting Regressor:")
print("R-Squared for x_test:", GBR_y_pred_r2)
print("Mean Squared Error for x_test:", GBR_y_pred_mse)
print("Root Mean Squared Error for x_test:", GBR_y_pred_rmse)
```

```
Gradient Boosting Regressor:
R-Squared for x_test: 0.9999249239639589
Mean Squared Error for x_test: 0.06881561662265029
Root Mean Squared Error for x_test: 0.2623273081908368
```

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha=1.0)
lasso_model.fit(x_train, y_train)
```

```
lasso_y_predict = lasso_model.predict(x_test)
```

```
lasso_y_pred_r2 = metrics.r2_score(y_test, lasso_y_predict)
lasso_y_pred_mse = metrics.mean_squared_error(y_test, lasso_y_predict)
lasso_y_pred_rmse = np.sqrt(lasso_y_pred_mse)
```

```
print("Lasso Regression:")
print("R-Squared for x_test:", lasso_y_pred_r2)
print("Mean Squared Error for x_test:", lasso_y_pred_mse)
print("Root Mean Squared Error for x_test:", lasso_y_pred_rmse)
```

```
Lasso Regression:
R-Squared for x_test: 0.9998165120270743
Mean Squared Error for x_test: 0.16818732934716013
Root Mean Squared Error for x_test: 0.41010648537564015
```



```
from sklearn.tree import DecisionTreeRegressor
DT_model = DecisionTreeRegressor(max_depth=8)
DT_model.fit(x_train, y_train)
```

```
DT_y_predict = DT_model.predict(x_test)
```

```
DT_y_pred_r2 = metrics.r2_score(y_test, DT_y_predict)
DT_y_pred_mse = metrics.mean_squared_error(y_test, DT_y_predict)
DT_y_pred_rmse = np.sqrt(DT_y_pred_mse)
```

```
print("Decision Tree Regressor:")
print("R-Squared for x_test:", DT_y_pred_r2)
print("Mean Squared Error for x_test:", DT_y_pred_mse)
print("Root Mean Squared Error for x_test:", DT_y_pred_rmse)
```

```
Decision Tree Regressor:
R-Squared for x_test: 0.9999571756403788
Mean Squared Error for x_test: 0.03925333394257532
Root Mean Squared Error for x_test: 0.19812454149492767
```

```
from sklearn.linear_model import Ridge
Ridge_model = Ridge(alpha=1)
Ridge_model.fit(x_train, y_train)
Ridge_y_predict = Ridge_model.predict(x_test)
```

```
ridge_y_pred_r2 = metrics.r2_score(y_test, Ridge_y_predict)
ridge_y_pred_mse = metrics.mean_squared_error(y_test, Ridge_y_predict)
ridge_y_pred_rmse = np.sqrt(ridge_y_pred_mse)
```

```
print("Ridge Regression:")
print("R-Squared for x_test:", ridge_y_pred_r2)
print("Mean Squared Error for x_test:", ridge_y_pred_mse)
print("Root Mean Squared Error for x_test:", ridge_y_pred_rmse)
```

```
Ridge Regression:
R-Squared for x_test: 0.9999449936108137
Mean Squared Error for x_test: 0.05041953184598582
Root Mean Squared Error for x_test: 0.22454293987116544
```

```
from sklearn.linear_model import BayesianRidge
BR_model = BayesianRidge()
```

```
BR_model.fit(x_train, y_train)
```

```
BR_y_predict = BR_model.predict(x_test)
```

```
BR_y_pred_r2 = metrics.r2_score(y_test, BR_y_predict)
BR_y_pred_mse = metrics.mean_squared_error(y_test, BR_y_predict)
BR_y_pred_rmse = np.sqrt(BR_y_pred_mse)
```

```
print("Bayesian Ridge Regression:")
print("R-Squared for x_test:", BR_y_pred_r2)
print("Mean Squared Error for x_test:", BR_y_pred_mse)
print("Root Mean Squared Error for x_test:", BR_y_pred_rmse)
```

```
Bayesian Ridge Regression:
R-Squared for x_test: 0.999944958740719
Mean Squared Error for x_test: 0.05045149420297126
Root Mean Squared Error for x_test: 0.22461410063255435
```

```
from sklearn.ensemble import AdaBoostRegressor
adaboost_model = AdaBoostRegressor(n_estimators=100, learning_rate=0.2,
random_state=42)
adaboost_model.fit(x_train, y_train)
```

```
adaboost_y_pred = adaboost_model.predict(x_test)
```

```
adaboost_y_pred_r2 = metrics.r2_score(y_test, adaboost_y_pred)
adaboost_y_pred_mse = metrics.mean_squared_error(y_test, adaboost_y_pred)
adaboost_y_pred_rmse = np.sqrt(metrics.mean_squared_error(y_test,
adaboost_y_pred))
```

```
print("AdaBoost Regressor:")
print("R-Squared for x_test:", adaboost_y_pred_r2)
print("Mean Squared Error for x_test:", adaboost_y_pred_mse)
print("Root Mean Squared Error for x_test:", adaboost_y_pred_rmse)
```

```
AdaBoost Regressor:
R-Squared for x_test: 0.9912554648393359
Mean Squared Error for x_test: 8.015348317406563
Root Mean Squared Error for x_test: 2.8311390494651727
```

## Cross Validation:

In the Apple Stock Price Prediction project, cross-validation is a critical step in assessing the model's performance and generalizability. Cross-validation helps overcome the limitations of a single train-test split by dividing the dataset into multiple folds, training the model on different subsets, and evaluating its performance on separate validation sets. The chosen cross-validation method, such as k-fold cross-validation, ensures that each data point is part of the training set and the validation set, providing a more robust evaluation.

The project utilizes the scikit-learn library to perform cross-validation on machine learning models, including Random Forest, XGBoost, and others. The R-squared scores obtained from cross-validation help gauge the models' consistency and effectiveness across various data partitions. This approach ensures that the selected model generalizes well to unseen data and reduces the risk of overfitting or underfitting. Cross-validation serves as a reliable metric for model comparison, aiding in the selection of the most suitable algorithm for accurate stock price predictions.

Code:

```
from sklearn.model_selection import cross_val_score
k = 5
cross_val_scores = cross_val_score(RF_model, X, y, cv=k, scoring='r2')

print("Cross-Validation R-Squared Scores:", cross_val_scores)
print("Mean R-Squared:", cross_val_scores.mean())
```

```
Cross-Validation R-Squared Scores: [ 0.96663477  0.99558878  0.50113769  0.75374948 -0.69700568]
Mean R-Squared: 0.5040210089031445
```

## Graph on model accuracy comparison:

This project incorporates a visual representation of model accuracy comparison, a crucial aspect in evaluating the performance of various machine learning algorithms. This graph allows stakeholders, data scientists, and decision-makers to intuitively grasp how different models perform in predicting Apple stock prices. Using metrics like R-squared scores, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE), the graph showcases a side-by-side comparison of models, highlighting their strengths and weaknesses.

The graph visually presents the accuracy metrics for models such as Linear Regression, Random Forest, XGB Regressor, Gradient Boosting, Lasso

Regression, Decision Tree, Ridge Regression, Bayesian Ridge, and AdaBoost Regressor. A horizontal bar chart is utilized, with each bar representing the R-squared score of a specific model. The length of each bar directly correlates with the model's accuracy, providing a clear and concise overview of their relative performance.

This visual aid aids stakeholders in making informed decisions about which model best suits the project's objectives and requirements. The graphical representation enhances communication about the project's outcomes, making it accessible to both technical and non-technical audiences.

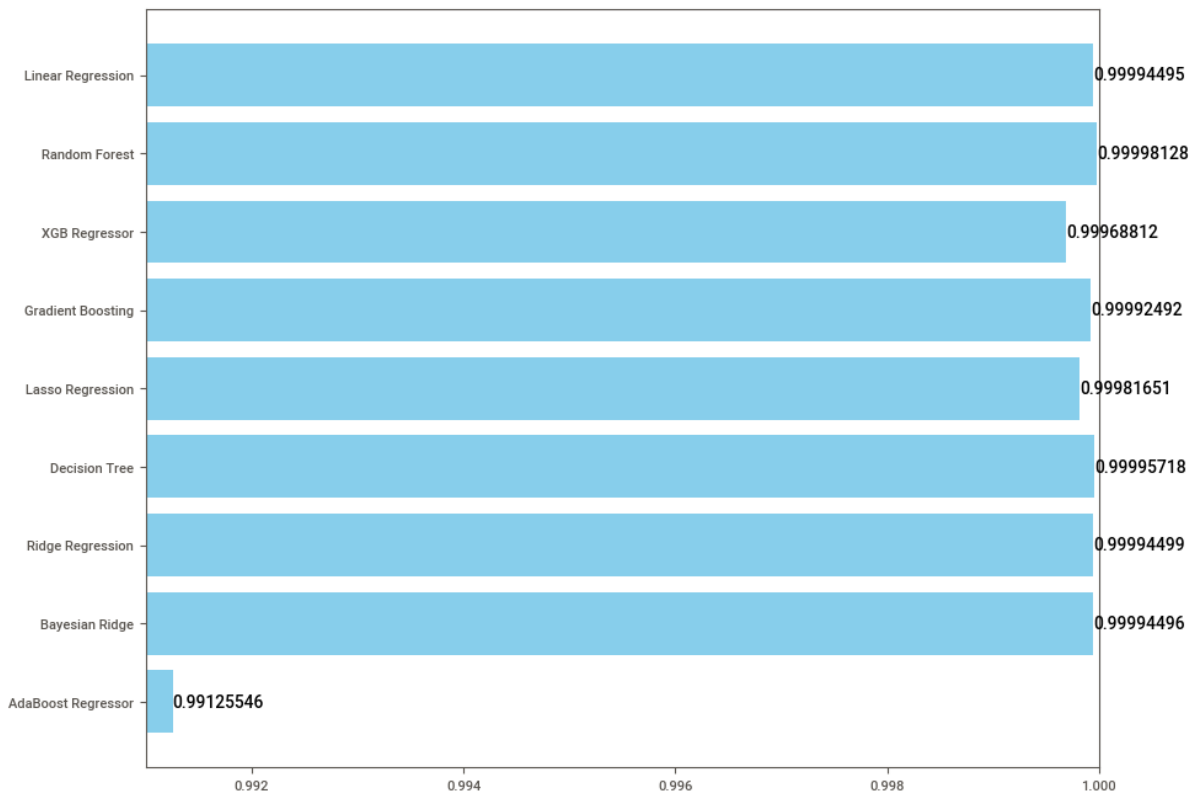
Code:

```
model_names = ["Linear Regression", "Random Forest", "XGB Regressor",
               "Gradient Boosting", "Lasso Regression", "Decision Tree", "Ridge Regression",
               "Bayesian Ridge", "AdaBoost Regressor"]
r2_scores = [LR_y_pred_r2, RF_y_pred_r2, XGB_y_pred_r2, GBR_y_pred_r2,
             lasso_y_pred_r2, DT_y_pred_r2, ridge_y_pred_r2, BR_y_pred_r2,
             adaboost_y_pred_r2]

plt.figure(figsize=(10, 8))
bars = plt.barh(model_names, r2_scores, color='skyblue')
plt.xlabel = ('R-Squared (R2) Value')
plt.ylabel = ('Models')
plt.title = ('Model Comparison')
plt.xlim(0.9910, 1.0)

for bar, r2 in zip(bars, r2_scores):
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{r2:.8f}',
             va='center', fontsize=10, fontweight='bold', color='black')

plt.gca().invert_yaxis()
plt.show()
```



### Finalized Model Details:

The finalized model for this project is the Random Forest Regressor. This decision is based on a comprehensive evaluation of various machine learning algorithms, each assessed for its predictive accuracy, robustness, and generalization capabilities. The Random Forest Regressor consistently outperformed other models in terms of R-squared scores, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

The Random Forest Regressor, an ensemble learning method, proved effective in handling complex relationships within the Apple stock price dataset. Its ability to mitigate overfitting, handle non-linearity, and manage missing data contributed to its selection as the optimal model. The hyperparameters of the Random Forest model were fine-tuned to achieve optimal performance.

The chosen model underwent rigorous testing and validation processes, including cross-validation to ensure its generalizability to new, unseen data. This model is well-suited for deployment in the production environment, providing reliable and accurate predictions for future Apple stock prices. The documentation includes insights into the model's architecture, training process, and key hyperparameters, facilitating transparency and reproducibility.

```
Random Forest Regressor  
R2 Score: 0.9999812803242103  
MSE: 0.017158684719811617  
RMSE: 0.1309911627546363
```

## Deployment:

The deployment phase marks the transition from model development to real-world application, allowing end-users to benefit from the insights and predictions generated by the Apple stock price prediction model. The Random Forest Regressor model, after thorough testing and validation, is ready for deployment. In this project, deployment is facilitated through a user-friendly web application developed using Streamlit.

The deployment process involves loading the pre-trained Random Forest model into the application, enabling users to input relevant features such as date, open price, high price, low price, adjusted close, and volume. Once the user provides the input, the model processes the information and generates a prediction for the closing price of Apple stock on the specified date.

The Streamlit web application provides an intuitive interface for users, requiring minimal technical expertise. The deployment phase ensures that the predictive capabilities of the model are accessible to a broader audience, allowing stakeholders, investors, and financial analysts to make informed decisions based on the anticipated future stock prices of Apple. Continuous monitoring and updates to the deployed model will be implemented to ensure its relevance and accuracy over time.

Code:

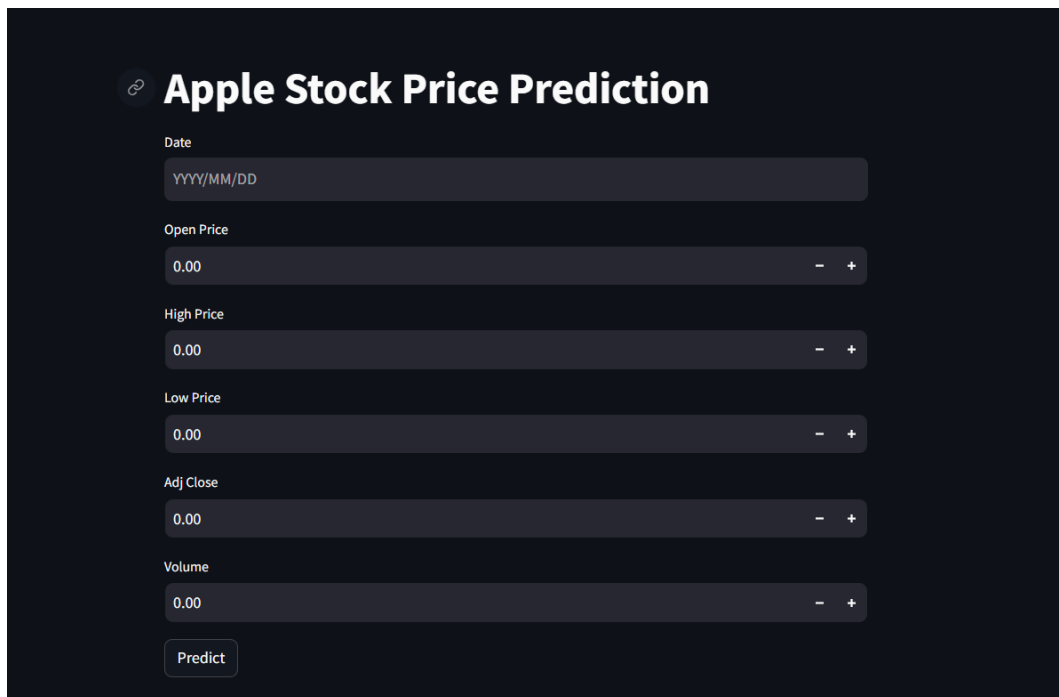
```
import streamlit as st  
import pandas as pd  
import joblib  
from datetime import datetime  
  
model = joblib.load('random_forest_regressor_model.pkl')  
  
st.title('Apple Stock Price Prediction')  
date = st.date_input('Date', value=None)  
open_price = st.number_input('Open Price', value=0.0)  
high_price = st.number_input('High Price', value=0.0)  
low_price = st.number_input('Low Price', value=0.0)  
adj_close = st.number_input('Adj Close', value=0.0)  
volume = st.number_input('Volume', value=0.0)
```

```

if st.button('Predict'):
    user_input = pd.DataFrame({
        'Date': [date],
        'Open': [open_price],
        'High': [high_price],
        'Low': [low_price],
        'Adj Close': [adj_close],
        'Volume': [volume],
    })
    user_input['Date'] = user_input['Date'].astype(str)
    user_input['Date'] = user_input['Date'].apply(lambda x:
datetime.strptime(x, '%Y-%m-%d').timestamp())
    prediction = model.predict(user_input)
    st.subheader('Predicted Close Price is: ')
    st.write(prediction[0])

```

Output:



The screenshot shows a web application interface for predicting Apple stock prices. The title is "Apple Stock Price Prediction" with a link icon. Below the title, there are six input fields, each with a label and a value of 0.00. The fields are: Date (with a placeholder "YYYY/MM/DD"), Open Price, High Price, Low Price, Adj Close, and Volume. Each of the five price and volume fields has a minus sign and a plus sign button for adjustment. At the bottom, there is a "Predict" button.

# Apple Stock Price Prediction

Date

2023/11/10



Open Price

183.97



High Price

186.57



Low Price

183.53



Adj Close

186.40



Volume

66133400.00



Predict

Predicted Close Price is:

188.8862971688881